PCT/JP03/13302

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００２年１０月１８日

出願番号
Application Number: 特願２００２－３０５０１０

[ST. 10/C]: [ＪＰ２００２－３０５０１０]

出願人
Applicant(s): 株式会社アドバンテスト

２００３年１１月２０日

特許庁長官
Commissioner,
Japan Patent Office

今井康夫

出証番号　出証特２００３－３０９５９４

【書類名】　特許願

【整理番号】　PAVA-14158

【提出日】　平成14年10月18日

【あて先】　特許庁長官殿

【国際特許分類】　G06F　9/45

G06F 11/28　350

G06F 11/22　320

【発明者】

【住所又は居所】　東京都練馬区旭町１丁目３２番１号　株式会社アドバンテスト内

【氏名】　前田　裕紀

【特許出願人】

【識別番号】　390005175

【氏名又は名称】　株式会社アドバンテスト

【代理人】

【識別番号】　100089118

【弁理士】

【氏名又は名称】　酒井　宏明

【手数料の表示】

【予納台帳番号】　036711

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　　明細書

【発明の名称】　プログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法

【特許請求の範囲】

【請求項１】　汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのプログラム開発支援装置において、

前記独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイル手段と、

前記異種言語混在ソースプログラム内の前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイル手段と、

前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出し、抽出した独自仕様言語ソースプログラムを指定して前記独自仕様言語コンパイル手段を実行させるとともに、前記異種言語混在ソースプログラムを指定して前記汎用言語コンパイル手段を実行させ、得られた独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを結合してオブジェクトファイルを生成する統合コンパイル手段と、

前記統合コンパイル手段によって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンク手段と、

を備えたことを特徴とするプログラム開発支援装置。

【請求項２】　前記統合コンパイル手段は、前記オブジェクトファイルに、前記独自仕様言語オブジェクトコードおよび／または前記汎用言語オブジェクトコードのコード位置情報を付加することを特徴とする請求項１に記載のプログラム開発支援装置。

【請求項３】　前記プログラムファイルを前記プログラム実行装置に転送するプログラム転送手段を備えたことを特徴とする請求項１または２に記載のプログラム開発支援装置。

【請求項４】　前記プログラム実行装置に対して該プログラム実行装置に転送されたプログラムファイルを実行させる指示を与えるプログラム実行装置コントロール手段を備えたことを特徴とする請求項３に記載のプログラム開発支援装置。

【請求項５】　前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定手段と、

前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガを起動することを特徴とする請求項１～４のいずれか一つに記載のプログラム開発支援装置。

【請求項６】　前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示することを特徴とする請求項５に記載のプログラム開発支援装置。

【請求項７】　前記汎用言語はＣ言語であり、前記独自仕様言語ソースプログラムは、前記汎用言語ソースプログラムのプリプロセッサコマンドによって該汎用言語ソースプログラム内に記述されたことを特徴とする請求項１～６のいずれか一つに記載のプログラム開発支援装置。

【請求項８】　前記プリプロセッサコマンドは、＃ｐｒａｇｍａであることを特徴とする請求項７に記載のプログラム開発支援装置。

【請求項９】　前記プログラム実行装置は、半導体試験装置であることを特徴とする請求項１～８のいずれか一つに記載のプログラム開発支援装置。

【請求項１０】　汎用言語ソースプログラムのオブジェクトコードと独自仕様言語ソースプログラムのオブジェクトコードが混在したプログラムファイルを実行するプログラム実行装置において、

前記プログラムファイルの実行開始時に、前記汎用言語ソースプログラムのオブジェクトコードと前記独自仕様言語ソースプログラムのオブジェクトコードをメモリにロードすることを特徴とするプログラム実行装置。

【請求項１１】　前記プログラム実行装置は半導体試験装置であることを特徴とする請求項１０に記載のプログラム実行装置。

【請求項１２】　汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのコンパイル方法において、

前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出する独自仕様言語ソースプログラム抽出ステップと、

抽出された独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイルステップと、

前記異種言語混在ソースプログラムのうちの前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイルステップと、

前記独自仕様言語オブジェクトコードと前記汎用言語オブジェクトコードを結合してオブジェクトファイルを生成するオブジェクトファイル生成ステップと、

前記オブジェクトファイル生成ステップによって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンクステップと、

を含んだことを特徴とするコンパイル方法。

【請求項１３】　汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから作成された所定のプログラム実行装置上で実行可能なプログラムファイルをデバッグするためのデバッグ方法において、

前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定ステップと、

前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプロ

グラムに属する場合に独自仕様言語デバッガを起動するデバッガ起動ステップと、

前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示するデバッグ情報表示ステップと、

を含んだことを特徴とするデバッグ方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、半導体試験装置等の被制御装置に対する制御コマンドが記述された独自仕様言語プログラムと、独自仕様言語プログラムの実行ステップや独自仕様言語プログラムから得られたデータの処理ステップ等が記述された汎用言語プログラムとを開発するためのプログラム開発支援装置、それらプログラムを実行するプログラム実行装置、それらプログラムのコンパイル方法およびデバッグ方法に関し、特に、上記した独自仕様言語プログラムと汎用言語プログラムとが一つのファイル内に混在して記述された異種言語混在プログラムを開発するためのプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法に関する。

【０００２】

【従来の技術】

膨大かつ多様な信号処理が求められる測定装置や通信装置などの電子機器の多くには、高性能なプロセッサが搭載されている。それに伴い、プロセッサ上で実行されるプログラム（ファームウェア）もまた、複雑でサイズが大きくなる傾向にある。よって、必然と、これら電子機器に記録されているプログラムは、未知のバグを含んでいたり、機能の追加や改善を求められることが多くなっている。

【０００３】

そのため、このような電子機器は、動作の設定／監視や上記したプログラムのアップデートを可能にするために、通信ケーブルを介して、外部のコンピュータと接続可能な形態をとっていることが多い。すなわち、この形態では、外部のコンピュータから電子機器のプロセッサに対し、制御コマンドや制御プログラム自

体を配信することが可能になっている。さらに換言すれば、外部のコンピュータ上で開発したプログラムのアルゴリズムや設定内容に従って、電子機器を操作することが可能になっている。

【０００４】

そのような電子機器のうちでも特に半導体試験装置は、多種多様かつ独自仕様の半導体デバイスに対してそれぞれ個別のデバイステストプログラムを用意する必要がある特殊な測定装置であり、半導体試験装置のプロセッサ上で実行させるプログラムを、利用者（半導体デバイス製造メーカ）自らが開発するという形態が定着している。

【０００５】

ところが、半導体試験装置を筆頭とした特殊用途の電子機器は、搭載されるプロセッサも独自仕様のものが多く、必然とそのプロセッサが理解できるバイナリファイルも独自仕様に従ったものとなっている。そのため、そのようなバイナリファイルの元となるソースファイルの作成に必要なプログラム言語の仕様や開発支援環境もまた特殊となり、プログラム開発者は、プログラム言語とともに開発支援環境の操作も習熟する必要があった。

【０００６】

このような背景から、近年においては、Ｃ言語やＪＡＶＡ（登録商標）等の汎用言語によって開発されたプログラムを実行することができる電子機器も登場してきているが、そのような電子機器を採用することは、過去の独自仕様のプログラム資産を活用することができないという新たな問題を生み出す。

【０００７】

そこで、現状では、データ処理やアルゴリズム全体を、Ｃ言語のような汎用言語プログラムで記述し、その汎用言語プログラム内から、電子機器に依存する独自仕様言語プログラムをサブルーチンとして呼び出すという形態がみられる。以下に、この形態におけるプログラムの開発と実行に関し、プログラムを実行する電子機器が半導体試験装置である場合を例に挙げて詳述する。

【０００８】

まず、半導体試験装置上でのプログラムの実行動作を理解するために、半導体

試験装置とそのプログラム開発環境について説明する。半導体試験装置は、半導体メモリ、ロジックＩＣ、リニアＩＣなどの半導体デバイスに対して所定の動作試験を行う特殊な測定装置であり、一般にその装置構成は、上記した半導体デバイスの種別ごとに異なっている。また、半導体試験装置への試験実行指示、試験結果取得およびプログラム開発は、通常、半導体試験装置に接続されたワークステーションによって行なわれ、半導体試験は、半導体試験装置とワークステーションとで構成される半導体試験システム（例えば、特許文献１参照）によって実現される。

【０００９】

図１０は、従来の半導体試験システムの概略構成を示すブロック図であり、特に、上記したように異なる装置構成の半導体試験装置間において共通する構成を示すものである。図１０において、半導体試験装置１００は、テスタプロセッサ１１０と、テスタ本体１２０と、テストヘッド１３０と、通信インターフェース１４０とを備えて構成されている。

【００１０】

テスタプロセッサ１１０は、テスタ本体１２０との間で、制御コマンドの送信や試験データの送受信を行う手段であり、テスタ本体１２０の制御や後述するワークステーションとの間の通信を行うコントローラである。特に、テスタプロセッサ１１０は、内部に搭載したメモリ（図示略）に、ＯＳ（オペレーティングシステム）カーネル１１１を格納しており、デバイステストプログラムの起動や監視、メモリ管理などを行うとともに、同じくメモリに格納された通信バスドライバ１１２やテスタバスドライバ１１３を介して、通信インターフェース１４０の監視／制御、テスタ本体１２０の制御、試験データの送受信を行う。

【００１１】

ここで、デバイステストプログラムは、上記した汎用言語プログラム１１４と独自仕様言語プログラム１１７とで構成されており、それら全体で、被測定デバイス１３１に対する機能試験やＤＣパラメトリック試験等の各種の試験を実施する手順等を規定している。汎用言語プログラム１１４は、試験結果として取得した各種データを処理するコマンドを含んだステートメントと、デバイステストプ

ログラム全体をどのように実行するかを示すコマンドを含んだステートメントとによって構成されており、ＯＳカーネル１１１上で直接実行可能なバイナリファイルである。

【００１２】

一方、独自仕様言語プログラム１１７は、テスタ本体１２０を制御するためのコマンドで構成されるオブジェクトファイルである。但し、そのオブジェクトファイルは、過去の資産である独自仕様言語プログラムと同様に、独自仕様言語プログラム１１７に最適化されたカーネル上でのみ直接実行可能なバイナリファイルである。そのため、独自仕様言語プログラム１１７をＯＳカーネル１１１上で実行させるにあたっては、実行用エミュレータ１１５による解釈処理を要する。また、独自仕様言語プログラム１１７は、後述するワークステーション２００に対してのディスク・アクセス、キー入力、ディスプレイ表示といった入出力コマンドも含んでおり、それら入出力コマンドの実行にあたっては、実行用エミュレータ１１５による解釈に加え、さらにＩＯ制御用エミュレータ１１６による解釈処理を要する。

【００１３】

テスタ本体１２０は、テスタプロセッサ１１０から送信される制御コマンドに従って、テストヘッド１３０に実装された被測定デバイス１３１に対し、機能試験やＤＣパラメトリック試験、ＲＦ試験（高周波試験）等の各種の試験を行う手段であり、レジスタ１２１、メモリ１２２、試験信号送受信部１２３を備えて構成されている。レジスタ１２１は、テスタプロセッサ１１０内の上記テスタバスドライバ１１３との間で送受信される各種のデータを格納し、格納されたデータは、直接またはメモリ１２２を介して試験信号送受信部１２３に渡される。

【００１４】

また、試験信号送受信部１２３から出力されるデータは、一旦レジスタ１２１やメモリ１２２に格納された後、レジスタ１２１を介してテスタプロセッサ１１０内の上記テスタバスドライバ１１３に渡される。試験信号送受信部１２３は、パターン発生器やタイミング発生器、ＤＣユニット等の種々の試験ユニットから構成され、これら試験ユニットによって生成された試験信号を被測定デバイス１

３１に入力するとともに、被測定デバイス１３１の出力ピンに現れるデータを取得する。

【００１５】

図１１は、上記したワークステーション２００の概略構成を示すブロック図である。ワークステーション２００は、半導体試験装置１００内のテスタプロセッサ１１０に対してプログラムの転送や実行指示を行うコンソール端末の役割と、上記した汎用言語プログラム１１４や独自仕様言語プログラム１１７の開発を支援するプログラム開発支援装置の役割とを担う。図１１において、ワークステーション２００は、プロセッサ２２０と、通信インターフェース２４１と、ハードディスク装置２４２と、マウス２４３と、キーボード２４４と、ディスプレイ装置２４５とを備えて構成されている。

【００１６】

プロセッサ２２０は、内部に搭載したメモリ（図示略）に、ＯＳ（オペレーティングシステム）カーネル２２１を格納しており、種々のプログラムの起動や監視、メモリ管理などを行うとともに、同じくメモリに格納される通信バスドライバ２２３、ハードディスクドライバ２２４、マウスドライバ２２５、キーボードドライバ２２６、ディスプレイドライバ２２７を介して、通信インターフェース２４１の監視や制御、ハードディスク装置２４２との間でのプログラムやデータの読み込み／書き込み、マウス２４３やキーボード２４４からの入力情報の取得、ディスプレイ装置２４５への表示情報の出力などを行う。ここで特に、通信インターフェース２４１は、通信ケーブル（図示せず）を介して、図１０に示した通信インターフェース１４０と接続されており、ワークステーション２００と半導体試験装置１００との間の通信を可能としている。

【００１７】

また、ＯＳカーネル２２１は、ＧＵＩ（Ｇｒａｐｈｉｃａｌ　Ｕｓｅｒ　Ｉｎｔｅｒｆａｃｅ）処理部２２２を有しており、図示するエディタ２２８、汎用言語コンパイラ２２９、リンカ２３３、汎用言語デバッガ２３１、独自仕様言語コンパイラ２３０、独自仕様言語デバッガ２３２などの各種プログラムは、ディスプレイ装置２４５上に表示されるウィンドウ画面単位で実行可能である。なお、

このワークステーション２００は、汎用的なコンピュータの装置構成と同等である。よって、上記した各種ドライバや各種プログラムは、通常はハードディスク装置２４２に記憶されており、ＯＳカーネル２２１に従って必要に応じて上記メモリに読み込まれて実行される。

【００１８】

次に、上記した半導体試験装置１００とワークステーション２００とで構成される半導体試験システムにおいて、デバイステストプログラムの開発と実行の流れについて説明する。図１２は、従来のデバイステストプログラムの開発と実行手順を示したフローチャートである。ここでは、デバイステストプログラムが、上記したように汎用言語プログラムと独自仕様言語プログラムとで構成されており、汎用言語プログラムとしてＣ言語を採用し、独自仕様言語プログラムとしてＡＴＬ（アドバンテスト社独自規格）を採用した場合を例に挙げる。

【００１９】

まず、プログラム開発者は、ワークステーション２００上においてエディタ２２８を起動させ、Ｃ言語によるソースプログラムを作成する（ステップＳ３０１）。このソースプログラムは、上述したように、デバイステストプログラム全体のアルゴリズムを記述するものであり、シーケンス処理の所望の位置で、ＡＴＬで記述されたオブジェクトプログラムを呼び出して実行したり、その実行によって得られた試験結果データを処理する手順を規定している。

【００２０】

Ｃ言語によるソースプログラムの作成を終えると、プログラム開発者は、Ｃコンパイラ（汎用言語コンパイラ２２９に相当）に対し、作成したソースプログラムのファイル（以下、必要なヘッダファイル等を含めてＣソースファイルと称する。）を指定してコンパイルを実行させる（ステップＳ３０２）。コンパイル処理では、まず構文チェックが行なわれ、構文エラーが生じた場合には、エディタ２２８でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合には、上記したＣソースファイルを機械語に翻訳したオブジェクトファイル（以下、Ｃオブジェクトファイルと称する。）が生成される。

【００２１】

ステップＳ３０１で作成された複数のＣソースファイルに対し、上記したステップＳ３０２が完了すると、プログラム開発者は、リンカ２３３に対し、生成された複数のＣオブジェクトファイルやその他に必要なライブラリファイルを指定してリンクを実行させる（ステップＳ３０３）。このリンクによって、半導体試験装置１００のテスタプロセッサ１１０上で直接実行可能な単一のＣオブジェクトファイルが生成される。

【００２２】

また、プログラム開発者は、Ｃ言語によるオブジェクトファイルの生成と並行して、ワークステーション２００上においてエディタ２２８を起動させ、ＡＴＬによるソースプログラムを作成する（ステップＳ４０１）。このソースプログラムは、上述したように、半導体試験装置１００を制御するための制御コマンドを記述するものである。

【００２３】

ＡＴＬによるソースプログラムの作成が終えると、プログラム開発者は、ＡＴＬコンパイラ（独自仕様言語コンパイラ２３０に相当）に対し、作成したソースプログラムのファイル（以下、ＡＴＬソースファイルと称する。）を指定してコンパイルを実行させる（ステップＳ４０２）。なお、このコンパイル処理でも、上記したステップＳ３０２と同様に、まず構文チェックが行なわれ、構文エラーが生じた場合には、エディタ２２８でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合には、上記したＡＴＬソースプログラムは、上記したＣオブジェクトファイルで表わされる機械語（ある特定のテスタプロセッサで理解できる機械語）とは異なる旧テスタプロセッサ仕様の機械語に翻訳され、オブジェクトファイル（以下、ＡＴＬオブジェクトファイルと称する。）が生成される。

【００２４】

このような手順によって、単一Ｃオブジェクトファイルとａｔｌオブジェクトファイル群が用意されると、プログラム開発者は、ワークステーション２００上において、半導体試験装置１００との通信を可能にするコントロールプログラムを起動し、そのコントロールプログラムを用いて単一Ｃオブジェクトファイルと

ＡＴＬオブジェクトファイル群を半導体試験装置１００のテスタプロセッサ１１０へと転送する（ステップＳ３０４，ステップＳ４０３）。

【００２５】

続いて、プログラム開発者は、上記したコントロールプログラムに対して、単一Ｃオブジェクトファイルの実行指示を与える（ステップＳ３０５）。これにより、半導体試験装置１００のテスタプロセッサ１１０は、単一Ｃオブジェクトファイルに記述されたアルゴリズムに従って、ＡＴＬオブジェクトファイルの実行→テスタ本体１２０の所望の試験ユニットの稼動→被測定デバイス１３１から得られた試験結果の取得→データ処理を繰り返す。この際、データ処理によって適宜加工等が行われた試験結果は、半導体試験装置１００の通信インターフェース１４０、通信ケーブル、ワークステーション２００の通信インターフェース２４１を介して、上記したコントロールプログラムで受け取ることができ、そのコントロールプログラムに割り当てられたウィンドウ画面上に表示される。

【００２６】

ここで、プログラム開発者は、試験結果が明らかに異常である場合等の不具合を発見した場合、デバイステストプログラムに論理的なエラーが含まれていると判断し、ワークステーション２００上において汎用言語デバッガ２３１を起動させ、Ｃソースファイル中の所定のステートメントにブレークポイントを設定する。そして、プログラム開発者がデバッグ開始を指示することにより、汎用言語デバッガ２３１は、再度、上記したステップＳ３０２～Ｓ３０５の手順によって単一Ｃオブジェクトファイルを実行し、実行されたステートメントが、設定したブレークポイントに達したことを検出すると、ブレークしたステートメントの段階で有効となっている変数を表示する。プログラム開発者は、この変数の確認によって論理的なエラーを発見すると、エディタ２２８を起動させ、Ｃソースファイルを適宜修正して上記したステップＳ３０２～Ｓ３０５の手順を繰り返す。

【００２７】

一方、プログラム開発者は、汎用言語デバッガ２３１によってＣソースファイルの論理的なエラーを発見できないときは、続いて、独自仕様言語デバッガ２３２を起動させ、ＡＴＬソースファイル中の所定のステートメントにブレークポイ

ントを設定し、上記同様のデバッグ処理を行う。

【００２８】

【特許文献１】

特開２００１－１９５２７５号公報

【００２９】

【発明が解決しようとする課題】

しかしながら、上記したように、被制御装置（上記例では半導体試験装置）上で実行させるプログラムを汎用言語と独自仕様言語のように異種言語で記述した場合には、独自仕様言語による過去のプログラム資産を活用できるものの、双方のプログラムの開発段階において、それぞれ個別のソースファイルを必要とする。上記した例では、ＣソースファイルとＡＴＬソースファイルとをそれぞれ別ファイルとして用意する必要がある。簡単に言えば、汎用言語で記述されたソースファイル中に独自仕様言語に作成されたオブジェクトファイルの呼び出しを埋め込んだ場合には、少なくとも２つのソースファイルが必要となる。特に、ある一つの汎用言語ソースファイルに対しては、ある特定の独自仕様言語ソースファイルが対応するため、これらファイルはひとまとめで管理しなければならない。

【００３０】

さらに、双方のソースファイルの内容から、関連する異種言語のソースファイルを特定することは困難であり、一度管理を誤ると、対応関係を再度見出すのに多くの時間と手間を要するという問題があった。このことから、エディタ上でのソースファイルの修正作業や他のソースファイルの部分的な流用については慎重にならざるを得ず、プログラム開発者にとってもミスを増やす原因となっていた。

【００３１】

また、一つの実行プログラムに対して、汎用言語ソースファイルと独自仕様言語ソースファイルとを用意することから、必然と、それらをコンパイルすることによって同じ数だけオブジェクトファイルが生成される。これは、上記した管理がさらに複雑になることを意味する。このように、従来では、一つの実行プログラムに対して異種言語の複数のソースファイルおよびオブジェクトファイルが存

在したため、ファイル管理が煩雑となり、プログラムの開発効率を低下させるという問題があった。

【００３２】

本発明は上記に鑑みてなされたものであって、汎用言語ソースファイルのプリプロセッサ記述部に独自仕様言語のソースファイルを埋め込むことによって、独自仕様言語のプログラムによる過去の資産を活用できるとともに、一つの実行ファイルに対して必要なソースファイルおよびオブジェクトファイルの数を大幅に減少させることのできるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法を提供することを目的とする。

【００３３】

【課題を解決するための手段】

上記目的を達成するために、請求項１にかかるプログラム開発支援装置は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのプログラム開発支援装置において、前記独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する独自仕様言語コンパイル手段（後述する独自仕様言語コンパイラ３０に相当）と、前記異種言語混在ソースプログラム内の前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイル手段（後述する汎用言語コンパイラ２９に相当）と、前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出し、抽出した独自仕様言語ソースプログラムを指定して前記独自仕様言語コンパイル手段を実行させるとともに、前記異種言語混在ソースプログラムを指定して前記汎用言語コンパイル手段を実行させ、得られた独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを結合してオブジェクトファイルを生成する統合コンパイル手段（後述する統合コンパイラ３４に相当）と、前記統合コンパイル手段によって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンク手段（後述するリンカ３３に相当）と、を備えたことを特徴としている。

【００３４】

この発明によれば、汎用言語プログラムで記述されるアルゴリズム内に所定のプログラム実行装置特有の独自仕様言語プログラムの命令を組み込んだ態様において、独自仕様言語プログラムのソース自体が汎用言語プログラムのソース内に埋め込まれているので、独自仕様言語プログラムと汎用言語プログラムを、ソースファイルのみでなく、コンパイルによって作成されるオブジェクトファイルの段階においても一つのファイルとして取り扱うことができる。

【００３５】

また、請求項２にかかるプログラム開発支援装置は、上記した発明において、前記統合コンパイル手段が、前記オブジェクトファイルに、前記独自仕様言語オブジェクトコードおよび／または前記汎用言語オブジェクトコードのコード位置情報を付加することを特徴としている。

【００３６】

この発明によれば、コード位置情報を参照して、オブジェクトファイルから、独自仕様言語オブジェクトコードと汎用言語オブジェクトコードを取り出すことができる。

【００３７】

また、請求項３にかかるプログラム開発支援装置は、上記した発明において、前記プログラムファイルを前記プログラム実行装置に転送するプログラム転送手段（後述するコントロールプログラムに相当する）を備えたことを特徴としている。

【００３８】

この発明によれば、所定のプログラム実行装置へのファームウェアアップデートを行うことができる。

【００３９】

また、請求項４にかかるプログラム開発支援装置は、上記した発明において、前記プログラム実行装置に対して該プログラム実行装置に転送されたプログラムファイルを実行させる指示を与えるプログラム実行装置コントロール手段（後述するコントロールプログラムに相当）を備えたことを特徴としている。

【００４０】

この発明によれば、所定のプログラム実行装置へのプログラム実行指示、デバッグ等を行うことができる。

【００４１】

また、請求項５にかかるプログラム開発支援装置は、上記した発明において、前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定手段（後述する統合デバッガ３５に相当）と、前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガ（後述する汎用言語デバッガ３１に相当）を起動し、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガ（後述する独自仕様言語デバッガ３２に相当）を起動することを特徴としている。

【００４２】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムに対してデバッグを行う場合にも、既存するそれぞれのデバッガを利用することができる。

【００４３】

また、請求項６にかかるプログラム開発支援装置は、上記した発明において、前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報を共通するウィンドウ画面に表示することを特徴としている。

【００４４】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムに対してデバッグを行う場合にも、共通のデバッグ環境を利用することができる。

【００４５】

また、請求項７にかかるプログラム開発支援装置は、上記した発明において、前記汎用言語はＣ言語であり、前記独自仕様言語ソースプログラムは、前記汎用

言語ソースプログラムのプリプロセッサコマンドによって該汎用言語ソースプログラム内に記述されたことを特徴としている。

【００４６】

また、請求項８にかかるプログラム開発支援装置は、上記した発明において、前記プリプロセッサコマンドは、＃ｐｒａｇｍａであることを特徴としている。

【００４７】

また、請求項９にかかるプログラム開発支援装置は、上記した発明において、前記プログラム実行装置は、半導体試験装置であることを特徴としている。

【００４８】

また、請求項１０にかかるプログラム実行装置は、汎用言語ソースプログラムのオブジェクトコードと独自仕様言語ソースプログラムのオブジェクトコードが混在したプログラムファイルを実行するプログラム実行装置（後述する半導体試験装置１１に相当）において、前記プログラムファイルの実行開始時に、前記汎用言語ソースプログラムのオブジェクトコードと前記独自仕様言語ソースプログラムのオブジェクトコードをメモリにロードすることを特徴としている。

【００４９】

この発明によれば、独自仕様言語ソースプログラムとの汎用言語ソースプログラムとが混在した異種言語混在プログラムを実行することができる。

【００５０】

また、請求項１１にかかるプログラム実行装置は、上記した発明において、該プログラム実行装置が半導体試験装置であることを特徴としている。

【００５１】

また、請求項１２にかかるコンパイル方法は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから所定のプログラム実行装置上で実行可能なプログラムファイルを作成するためのコンパイル方法において、前記異種言語混在ソースプログラムから前記独自仕様言語ソースプログラムを抽出する独自仕様言語ソースプログラム抽出ステップ（後述するステップＳ１２１に相当）と、抽出された独自仕様言語ソースプログラムをコンパイルして独自仕様言語オブジェクトコードを生成する

独自仕様言語コンパイルステップ（後述するステップＳ１２３に相当）と、前記異種言語混在ソースプログラムのうちの前記汎用言語ソースプログラム記述部分をコンパイルして汎用言語オブジェクトコードを生成する汎用言語コンパイルステップ（後述するステップＳ１２２に相当）と、前記独自仕様言語オブジェクトコードと前記汎用言語オブジェクトコードを結合してオブジェクトファイルを生成するオブジェクトファイル生成ステップ（後述するステップＳ１２４に相当）と、前記オブジェクトファイル生成ステップによって生成された少なくとも一つのオブジェクトファイルから前記プログラムファイルを生成するリンクステップ（後述するステップＳ１３０に相当）と、を含んだことを特徴としている。

【００５２】

また、請求項１３にかかるデバッグ方法は、汎用言語ソースプログラムの所定領域に独自仕様言語ソースプログラムが記述された構成の異種言語混在ソースプログラムから作成された所定のプログラム実行装置上で実行可能なプログラムファイルをデバッグするためのデバッグ方法において、前記異種言語混在ソースプログラム内のステートメントにブレークポイントを設定するブレークポイント設定ステップと、前記プログラムファイルの実行時に前記ブレークポイントで該プログラムファイルを停止させるとともに、停止したプログラムファイルの前記ステートメントが前記汎用言語ソースプログラムに属する場合に汎用言語デバッガを起動し（後述するステップＳ２０３に相当）、停止したプログラムファイルの前記ステートメントが前記独自仕様言語ソースプログラムに属する場合に独自仕様言語デバッガを起動する（後述するステップＳ２０６に相当）デバッガ起動ステップと、前記汎用言語デバッガと前記独自仕様言語デバッガとから得られたデバッグ情報（後述するステップＳ２０４，ステップＳ２０７に相当）を共通するウィンドウ画面に表示するデバッグ情報表示ステップ（後述するステップＳ２０５に相当）と、を含んだことを特徴としている。

【００５３】

【発明の実施の形態】

以下に、本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法の実施の形態を図面に基づいて詳細に説明する。

なお、この実施の形態により本発明が限定されるものではない。

【００５４】

この実施の形態では、本発明の特徴の理解を容易にするために、上述した従来技術の説明と同様に、本発明を、半導体試験装置とワークステーションとから構成される半導体試験システムに適用した場合について説明する。具体的には、本発明にかかるプログラム開発支援装置が、半導体試験システムのワークステーションに相当し、本発明にかかるプログラム実行装置が半導体試験システムの半導体試験装置に相当し、本発明にかかるコンパイル方法およびデバッグ方法が上記半導体試験システム上でのコンパイル方法およびデバッグ方法に相当する。

【００５５】

図１は、本実施の形態にかかる半導体試験システムを示すブロック図である。図１に示す半導体試験システムは、通信ケーブルで接続されたワークステーション１０と半導体試験装置１１とを備えて構成される。

【００５６】

ワークステーション１０の基本構成は、図１１に示した従来の半導体試験システムのワークステーション２００と変わることはなく、図１中のプロセッサ２０、通信インターフェース４１、ハードディスク装置４２と、マウス４３と、キーボード４４と、ディスプレイ装置４５は、それぞれ図１１に示したプロセッサ２０、通信インターフェース２４１、ハードディスク装置２４２、マウス２４３、キーボード２４４、ディスプレイ装置２４５に対応する。

【００５７】

また、プロセッサ２０内部において、メモリ（図示略）に格納されるＯＳカーネル２１、ＧＵＩ処理部２２、通信バスドライバ２３、ハードディスクドライバ２４、マウスドライバ２５、キーボードドライバ２６、ディスプレイドライバ２７、エディタ２８、汎用言語コンパイラ２９、独自仕様言語コンパイラ３０、汎用言語デバッガ３１、独自仕様言語デバッガ３２、リンカ３３もまた、それぞれ図１１に示したＯＳカーネル２２１、ＧＵＩ処理部２２２、通信バスドライバ２２３、ハードディスクドライバ２２４、マウスドライバ２２５、キーボードドライバ２２６、ディスプレイドライバ２２７、エディタ２２８、汎用言語コンパイ

ラ２２９、独自仕様言語コンパイラ２３０、汎用言語デバッガ２３１、独自仕様言語デバッガ２３２、リンカ２３３と同様に機能する。

【００５８】

ここで、本実施の形態で説明するワークステーション１０において、従来のワークステーション２００と異なるのは、上記した汎用言語コンパイラ２９と独自仕様言語コンパイラ３０の上位アプリケーションに位置する統合コンパイラ３４を有している点である。換言すれば、このワークステーション１０は、統合コンパイラ３４から、汎用言語コンパイラ２９と独自仕様言語コンパイラ３０をそれぞれ利用できることを特徴としている。

【００５９】

他にも、本実施の形態で説明するワークステーション１０では、従来のワークステーション２００と異なる点として、上記した汎用言語デバッガ３１と独自仕様言語デバッガ３２の上位アプリケーションに位置する統合デバッガ３５を有している。すなわち、上記した統合コンパイラ３４と同様に、このワークステーション１０は、統合デバッガ３５から、汎用言語デバッガ３１と独自仕様言語デバッガ３２をそれぞれ利用できることを特徴としている。

【００６０】

また、図１に示す半導体試験システムでは、ワークステーション１０に半導体試験装置１１が接続されているが、この半導体試験装置１１の内部構成については図１０に示した従来の半導体試験装置１００と同様であるので、ここではその説明を省略する。但し、この半導体試験装置１１は、ワークステーション１０において開発されるプログラムの態様によっては、テスタプロセッサの動作が従来とは一部異なる。その説明については後述する。

【００６１】

次に、この半導体試験システムにおいて、デバイステストプログラムの開発と実行の流れについて説明する。図２は、本実施の形態におけるデバイステストプログラムの開発と実行手順を示したフローチャートである。ここでは、図１２の説明と同様に、デバイステストプログラムが、汎用言語プログラムと独自仕様言語プログラムとで構成されており、汎用言語プログラムとしてＣ言語を採用し、

独自仕様言語プログラムとしてＡＴＬ（アドバンテスト社独自規格）を採用した場合を例に挙げる。

【００６２】

まず、プログラム開発者は、ワークステーション１０上においてエディタ２８を起動させ、ソースプログラムを作成する（ステップＳ１１０）。ここで、このソースプログラムは、汎用言語であるＣ言語で記述されるが、図１２で説明したＣソースファイルの内容とは異なり、プリプロセッサ記述部に、独自仕様言語であるＡＴＬソースファイルの内容を記述することを特徴としている。ここで、このソースプログラムをＣ＋ＡＴＬソースプログラムと称する。

【００６３】

図３は、Ｃ＋ＡＴＬソースプログラムの記述例を示す図である。なお、図３において、左端に配置された「数字：」は行番号を示しており、ここでは説明の便宜上用いるが、実際のプログラム動作においては無視される。以下のＣ＋ＡＴＬソースプログラムの内容の説明では、この行番号によって各ステートメントを参照する。

【００６４】

Ｃ言語コンパイラは、先頭が＃で続くステートメントをプリプロセッサコマンドであると認識する。図３に示すＣ＋ＡＴＬソースプログラムでは、行番号１の＃ｉｎｃｌｕｄｅ、行番号３，４，５の＃ｐｒａｇｍａがそのプリプロセッサコマンドに相当する。＃ｉｎｃｌｕｄｅは、ヘッダファイル「ＡＴ／ｈｙｂｒｉｄ．ｈ」の記述内容を単にその位置に展開するのみであり、行番号１０以降に続くメイン関数内において必要となる内容である。

【００６５】

一方の＃ｐｒａｇｍａは、Ｃ言語との全般的な互換性を保ちながら、マシンまたはＯＳに固有の機能を実現する特別なプリプロセッサコマンドである。よって、＃ｐｒａｇｍａは定義上、マシンまたはＯＳ固有であり、通常コンパイラごとに異なる。＃ｐｒａｇｍａは、通常、プリプロセッサに新しい機能を与えたり、インプリメントに依存する情報をコンパイラに伝えるといった利用方法が主であるが、本実施の形態で作成されるＣ＋ＡＴＬソースプログラムでは、＃ｐｒａｇ

ｍａによって渡されるトークンとして、独自仕様言語であるＡＴＬの記述を埋め込むことを特徴としている。なお、この＃ｐｒａｇｍａによって渡されるＡＴＬの記述の処理については後述する。

【００６６】

図３に示すＣ＋ＡＴＬソースプログラムにおいて、行番号１０～２８に記述されている内容は、従来の半導体試験システムのワークステーション上において作成される内容と同じものであり、その記述の中において、ＡＴＬオブジェクトファイルの呼び出しやデータ処理が規定されている。

【００６７】

Ｃ＋ＡＴＬソースプログラムの作成が終えると、プログラム開発者は、統合コンパイラ３４に対し、作成したＣ＋ＡＴＬソースプログラムのファイル（以下、必要なヘッダファイル等を含めてＣ＋ＡＴＬソースファイルと称する。）を指定してコンパイルを実行させる（ステップＳ１２０）。このコンパイルの実行に際しては、まず構文チェックが行なわれ、構文エラーが生じた場合には、エディタ２８でエラー箇所を修正し、再度コンパイルを実行する。エラーがない場合に初めて、オブジェクトファイルを生成するためのコンパイル処理が開始される。

【００６８】

図４は、統合コンパイラ３４によるコンパイル処理を説明するためのフローチャートである。統合コンパイラ３４は、ステップＳ１１０において作成されたＣ＋ＡＴＬソースファイルにおいて構文エラーを発見できなかったときは、続いて、そのＣ＋ＡＴＬソースファイルからＡＴＬ記述部を抽出し、ＡＴＬソースファイルを生成する（ステップＳ１２１）。図５は、このＡＴＬソースファイルの生成処理を説明するための説明図である。ＡＴＬソースファイルの生成処理は、上述したように、Ｃ＋ＡＴＬソースファイルのプリプロセッサ記述部から、＃ｐｒａｇｍａを識別し、＃ｐｒａｇｍａの後に続くトークンを解析することから始まる。図５に示す例では、＃ｐｒａｇｍａの直後のａｔｌが、その後に続く情報がＡＴＬ記述部であることを示すキーワードとなる。

【００６９】

図５の例を用いて具体的な説明を行うと、統合コンパイラ３４は、行番号３に

おいて、＃ｐｒａｇｍａ　ａｔｌを認識すると、その後に続くキーワードのｎａｍｅを取り出し、そのキーワードｎａｍｅに続くダブルクォーテーションで囲まれた記述、すなわちＳＡＭＰＬＥがプログラム名であると解釈する。この解釈によって、ＡＴＬソースファイルの先頭部に、ＰＲＯ　ＳＡＭＰＬＥが挿入される。さらに、行番号４において、＃ｐｒａｇｍａ　ａｔｌを認識すると、その後に続くキーワードのｓｏｃｋｅｔを取り出し、そのキーワードｓｏｃｋｅｔに続くダブルクォーテーションで囲まれた記述、すなわちＳＳＯＣが使用するＡＴＬのソケットプログラム名であると解釈する。この解釈によって、ＡＴＬソースファイル内のＰＲＯ　ＳＡＭＰＬＥの後にＳＳＯＣが挿入される。

【００７０】

さらに、統合コンパイラ３４は、行番号５において、＃ｐｒａｇｍａ　ａｔｌを認識すると、その後に続くキーワードのｐｒｏｃを取り出し、そのキーワードｐｒｏｃの直後の文字列とその後に続くダブルクォーテーションで囲まれた記述、すなわち、

Ｐ１（ＡＲＧ１，ＡＲＧ２（２））　”　|ＷＲＩＴＥ　”ＡＲＧ１＝”，ＡＲＧ１，／ＷＲＩＴＥ　”ＡＲＧ２＝”，ＡＲＧ２，／|　”

が関数定義であると解釈する。この解釈によって、ＡＴＬソースファイルに、

```
Ｐ１：ＡＲＧＵＭＥＮＴ（ＡＲＧ１，ＡＲＧ２（２））
    ＷＲＩＴＥ　”ＡＲＧ１＝”，ＡＲＧ１，／
    ＷＲＩＴＥ　”ＡＲＧ２＝”，ＡＲＧ２，／
    ＧＯＴＯ　ＣＯＮＴＩＮＵＥ
```

が追記される。

【００７１】

そして、統合コンパイラ３４は、Ｃ＋ＡＴＬソースファイルにおいて、＃ｐｒａｇｍａ　ａｔｌを発見することができないまま、Ｃ＋ＡＴＬソースファイルの最終行（行番号２８）に達すると、ＡＴＬソースファイルの最後にＥＮＤを挿入してそのＡＴＬソースファイルの作成を終える。

【００７２】

ＡＴＬソースファイルの作成が終わると、統合コンパイラ３４は、Ｃ＋ＡＴＬ

ソースファイルのＣ言語記述部のコンパイル、すなわちＣオブジェクトコードの生成を行う（ステップＳ１２２）。このコンパイルは、通常のＣコンパイラ（汎用言語コンパイラ２９に相当する。）によって処理され、上記した＃ｐｒａｇｍａの記述部分は無視される。なお、ここでは、統合コンパイラ３４がＣコンパイラを呼び出して処理させるとしているが、統合コンパイラ３４自体にＣコンパイラの機能を取り入れ、上記したＡＴＬソースファイルの生成と並行して、Ｃオブジェクトコードの生成を行ってもよい。この場合、図１に示した汎用言語コンパイラ２９は不要となる。

【００７３】

Ｃオブジェクトコードの生成が終わると、統合コンパイラ３４は、ステップＳ１２１において生成されたＡＴＬソースファイルのコンパイル、すなわちＡＴＬオブジェクトコードの生成を行う（ステップＳ１２３）。このコンパイルは、ＡＴＬコンパイラ（独自仕様言語コンパイラ３０に相当する。）の呼び出しによって実行され、図１２のステップＳ４０２と同様に、上記したＣオブジェクトコードで表わされる機械語（ある特定のテスタプロセッサで理解できる機械語）とは異なる旧テスタプロセッサ仕様の機械語への翻訳を行う。

【００７４】

ＡＴＬオブジェクトコードの生成が終わると、統合コンパイラ３４は、ステップＳ１２２において生成されたＣオブジェクトコードに、ステップＳ１２１において生成されたＡＴＬオブジェクトコードを結合し、さらにそのＡＴＬオブジェクトコードが格納された位置情報（ＡＴＬオブジェクトコード開始位置）を付加したオブジェクトファイル（以下、Ｃ＋ＡＴＬオブジェクトファイルと称する。）を生成する（ステップＳ１２４）。図６は、このＣ＋ＡＴＬオブジェクトファイルの構成を示す図である。図６に示すように、Ｃ＋ＡＴＬオブジェクトファイルは、Ｃオブジェクトコードに続いてＡＴＬオブジェクトコードが配置される。なお、同図において、ＡＴＬオブジェクトコードの位置情報等の付加情報の図示は省略されている。

【００７５】

なお、この統合コンパイラ３４によるコンパイル処理は、上記同様に作成され

た複数のＣ＋ＡＴＬソースファイルに対して行われ、これにより複数のＣ＋ＡＴＬオブジェクトファイルが用意される。このようにして統合コンパイラ３４によるコンパイル処理が終わると、プログラム開発者は、リンカ３３に対し、生成された複数のＣ＋ＡＴＬオブジェクトファイルやその他に必要なライブラリファイルを指定してリンクを実行させる（図２のステップＳ１３０）。

【００７６】

リンカ３３は、複数のＣ＋ＡＴＬオブジェクトファイルやその他に必要なライブラリファイルに加え、上記各Ｃ＋ＡＴＬオブジェクトファイルからＡＴＬオブジェクトコード部分をロードするためのロードプログラムを用意し、それらをリンクすることによって、半導体試験装置１１のテスタプロセッサ上で直接実行可能な単一オブジェクトファイルを生成する。

【００７７】

このような手順によって、単一オブジェクトファイルが用意されると、プログラム開発者は、ワークステーション１０上において、半導体試験装置１１との通信を可能にするコントロールプログラムを起動し、そのコントロールプログラムを用いて上記単一オブジェクトファイルを半導体試験装置１１のテスタプロセッサへと転送する（ステップＳ１４０）。

【００７８】

続いて、プログラム開発者は、上記したコントロールプログラムに対して、単一オブジェクトファイルの実行指示を与える（ステップＳ１５０）。これにより、半導体試験装置１１のテスタプロセッサは、まず、単一オブジェクトファイルに含まれているロードプログラムに従って、同単一オブジェクトファイル内に配置されているＣオブジェクトコードとＡＴＬオブジェクトコードをメモリ上にロードする。続いて、テスタプロセッサは、ロードされたＣオブジェクトコードに記述されたアルゴリズムに従って、同じくロードされているＡＴＬオブジェクトコードの実行→テスタ本体の所望の試験ユニットの稼動→被測定デバイスから得られた試験結果の取得→データ処理を繰り返す。この際、データ処理によって適宜加工等が行われた試験結果は、従来と同様に、半導体試験装置１１の通信インターフェース、通信ケーブル、ワークステーション１０の通信インターフェース

４１を介して、上記したコントロールプログラムで受け取ることができ、そのコントロールプログラムに割り当てられたウィンドウ画面上に表示される。

【００７９】

なお、ここでは、単一オブジェクトファイル内に、ロードプログラムが含まれているとしたが、半導体試験装置１１のテスタプロセッサ内において、あらかじめそのロードプログラムを読み込ませておき、ワークステーション１０からの実行指示に従って、まずこのロードプログラムが起動するようにすることもできる。

【００８０】

次に、本実施の形態における半導体試験システムのデバック処理について説明する。プログラム開発者は、上記したステップＳ１５０の実行によって得た試験結果が、明らかに異常である場合等の不具合を発見した場合、従来と同様、デバイステストプログラムに対してデバック処理を行う。そこでまず、プログラム開発者は、ワークステーション１０上において、統合デバッガ３５を起動させ、Ｃ＋ＡＴＬソースファイル中の所定のステートメントにブレークポイントを設定する。

【００８１】

そして、プログラム開発者がデバッグ開始を指示することによって、統合デバッガ３５は、再度、上記したステップＳ１２０～Ｓ１５０の手順によって単一オブジェクトファイルを実行し、実行されたステートメントが、設定したブレークポイントに達したことを検出すると、Ｃデバッガ（汎用言語デバッガ３１に相当する。）とＡＴＬデバッガ（独自仕様言語デバッガ３２に相当する。）のいずれかのデバッガを起動させるかのデバッガ選択ルーチンを実行する。

【００８２】

図７は、このデバッガ選択ルーチンを示すフローチャートである。統合デバッガ３５は、単一オブジェクトファイル中において順次実行されているステートメントがブレークポイントに到達すると、そのブレークポイントが設定されたステートメントを表示する（ステップＳ２０１）。そして、そのステートメントがＡＴＬオブジェクトコード部に相当するならば（ステップＳ２０２：Ｙｅｓ）、Ａ

ＴＬデバッガを起動し（ステップＳ２０６）、ＡＴＬデバッガからブレークしたステートメントに含まれる変数等のデバッグ情報を取得する（ステップＳ２０７）。なお、ＡＴＬデバッガは、上記したブレークポイント設定時に、統合デバッガ３５によってＡＴＬオブジェクトコード部のブレークポイント設定情報を取得している。

【００８３】

統合デバッガ３５は、ＡＴＬデバッガからデバッグ情報を取得すると、ブレーク時に有効となっている指定変数（シンボル）を表示する（ステップＳ２０５）。図８は、統合デバッガ３５の実行画面の例を示す図であり、特にＡＴＬ記述部内でブレークした状態を示している。図８において、統合デバッガ３５は、実行ウィンドウ５０内に、ウィンドウ表示に標準として付加されるタイトルバーやメニューバーに加え、ブレークポイント設定領域５１、ソース表示領域５２、シンボル表示領域５３を有している。ここでは、Ｃ＋ＡＴＬソースプログラム中のＡＴＬ記述部内でブレークした状態として、ブレークポイントが設定された行番号５のステートメントがソース表示領域５２内に表示されるとともに、シンボル表示領域５３内に、そのステートメントで使用されている変数と変数に格納された値が表示されている。

【００８４】

一方、統合デバッガ３５は、ブレークしたステートメントがＣオブジェクトコード部に相当するならば（ステップＳ２０２：Ｎｏ）、Ｃデバッガを起動し（ステップＳ２０３）、Ｃデバッガからブレークしたステートメントに含まれる変数等のデバッグ情報を取得する（ステップＳ２０４）。なお、Ｃデバッガは、上記したブレークポイント設定時に、統合デバッガ３５によってＣオブジェクトコード部のブレークポイント設定情報を取得している。

【００８５】

統合デバッガ３５は、Ｃデバッガからデバッグ情報を取得すると、ブレーク時に有効となっている指定変数（シンボル）を表示する（ステップＳ２０５）。図９は、統合デバッガ３５の実行画面の例を示す図であり、特にＣ言語記述部内でブレークした状態を示している。図９に示す統合デバッガ３５の実行ウィンドウ

５０は、図８と同様な構成であるが、ここでは、Ｃ＋ＡＴＬソースプログラム中のＣ言語記述部内でブレークした状態として、ブレークポイントが設定された行番号１５のステートメントがソース表示領域５２内に表示されるとともに、シンボル表示領域５３内に、そのステートメントで使用されている変数と変数に格納された値が表示されている。

【００８６】

プログラム開発者は、この統合デバッガ３５のウィンドウ画面上に表示された変数の値を確認することによって論理的なエラーを発見した場合、エディタ２８を起動させ、Ｃ＋ＡＴＬソースファイルを適宜修正して上記したステップＳ１２０～Ｓ１５０の手順を繰り返す。

【００８７】

以上に説明したとおり、実施の形態にかかる半導体試験システム、すなわち本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法によれば、独自仕様言語プログラムであるＡＴＬソース自体を汎用言語プログラムであるＣ言語ソース内に埋め込むことによって、従来別管理されていた双方のソースを一つのＣ＋ＡＴＬソースファイルとして取り扱うことかできるので、ファイル管理を楽にし、プログラムの開発効率を向上させることができる。

【００８８】

なお、本実施の形態では、本発明にかかるプログラム開発支援装置とプログラム実行装置をそれぞれワークステーションと半導体試験装置に適用した場合を例示したが、プログラム開発支援装置を汎用的なコンピュータシステムとし、プログラム実行装置をそのコンピュータシステムと通信可能な測定装置や制御装置に適用することができるのは言うまでもない。

【００８９】

【発明の効果】

以上に説明したように、本発明にかかるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法によれば、独自仕様言語プログラム自体を汎用言語プログラム内に埋め込むことで、従来別管理されていた双方の

ソースプログラムを、ソースファイルのみでなく、コンパイルによって作成されるオブジェクトファイルの段階においても一つのファイルとして取り扱うことができるので、ファイル管理を楽にし、プログラムの開発効率を向上させることができるという効果を奏する。

【図面の簡単な説明】

【図１】

実施の形態にかかる半導体試験システムの概略構成を示すブロック図である。

【図２】

デバイステストプログラムの開発と実行手順を示したフローチャートである。

【図３】

Ｃ＋ＡＴＬソースプログラムの記述例を示す図である。

【図４】

統合コンパイラによるコンパイル処理を説明するためのフローチャートである。

【図５】

ＡＴＬソースファイルの生成処理を説明するための説明図である。

【図６】

Ｃ＋ＡＴＬオブジェクトファイルの構成を示す図である。

【図７】

デバッガ選択ルーチンを示すフローチャートである。

【図８】

ＡＴＬ記述部内でブレークした状態の統合デバッガの実行画面の例を示す図である。

【図９】

Ｃ言語記述部内でブレークした状態の統合デバッガの実行画面の例を示す図である。

【図１０】

従来の半導体試験システムの概略構成を示すブロック図である。

【図１１】

従来の半導体試験システムのワークステーションの概略構成を示すブロック図である。

【図１２】

従来のデバイステストプログラムの開発と実行手順を示したフローチャートである。

【符号の説明】

１０，２００　ワークステーション

１１，１００　半導体試験装置

２０，２２０　プロセッサ

２１，１１１，２２１　ＯＳカーネル

２２，２２２　ＧＵＩ処理部

２３，１１２，２２３　通信バスドライバ

２４，２２４　ハードディスクドライバ

２５，２２５　マウスドライバ

２６，２２６　キーボードドライバ

２７，２２７　ディスプレイドライバ

２８，２２８　エディタ

２９，２２９　汎用言語コンパイラ

３０，２３０　独自仕様言語コンパイラ

３１，２３１　汎用言語デバッガ

３２，２３２　独自仕様言語デバッガ

３３，２３３　リンカ

３４　統合コンパイラ

３５　統合デバッガ

４１，１４０，２４１　通信インターフェース

４２，２４２　ハードディスク装置

４３，２４３　マウス

４４，２４４　キーボード

４５，２４５　ディスプレイ装置

５０　実行ウィンドウ

５１　ブレークポイント設定領域

５２　ソース表示領域

５３　シンボル表示領域

１１０　テスタプロセッサ

１１３　テスタバスドライバ

１１４　汎用言語プログラム

１１５　実行用エミュレータ

１１６　制御用エミュレータ

１１７　独自仕様言語プログラム

１２０　テスタ本体

１２１　レジスタ

１２２　メモリ

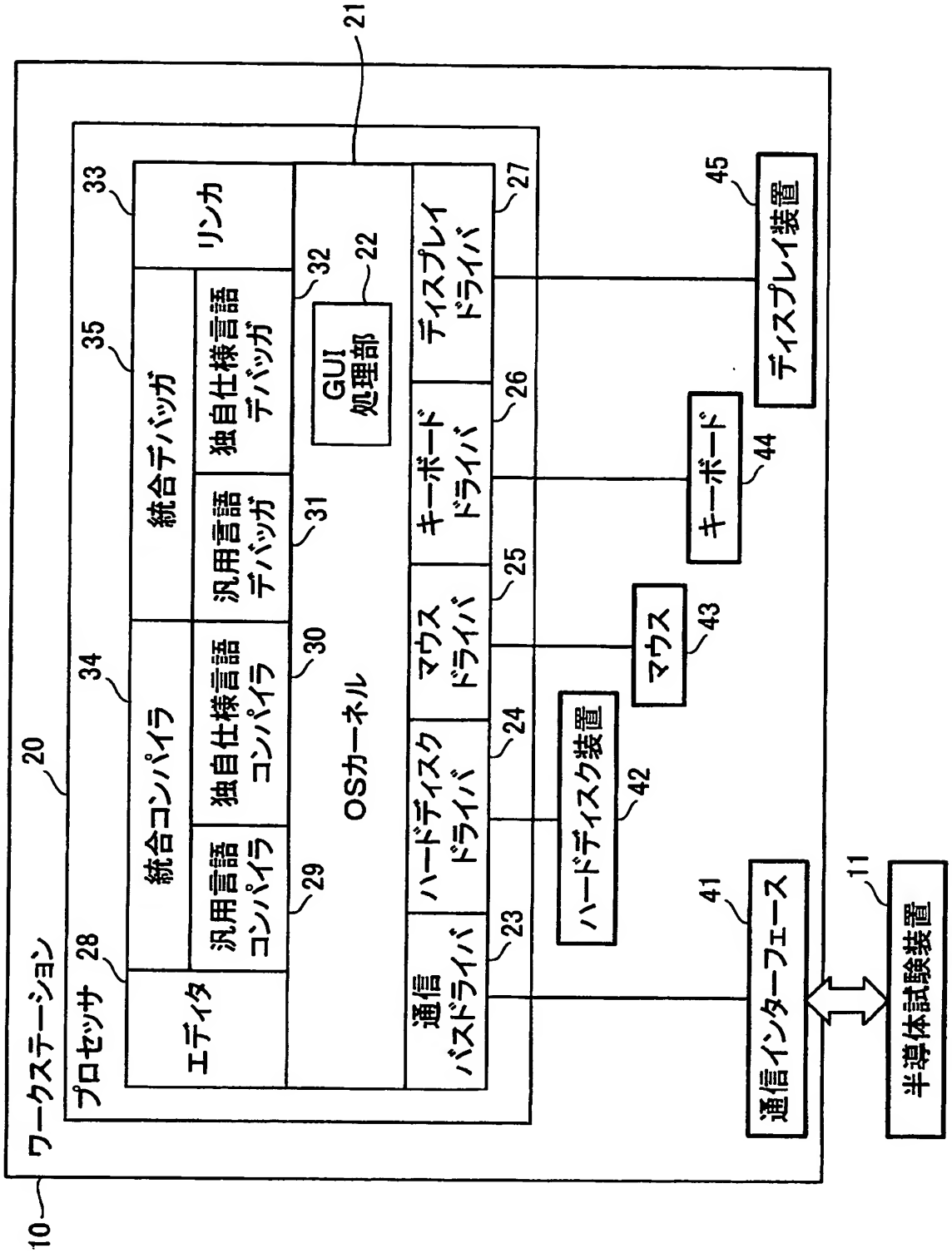
１２３　試験信号送受信部

１３０　テストヘッド

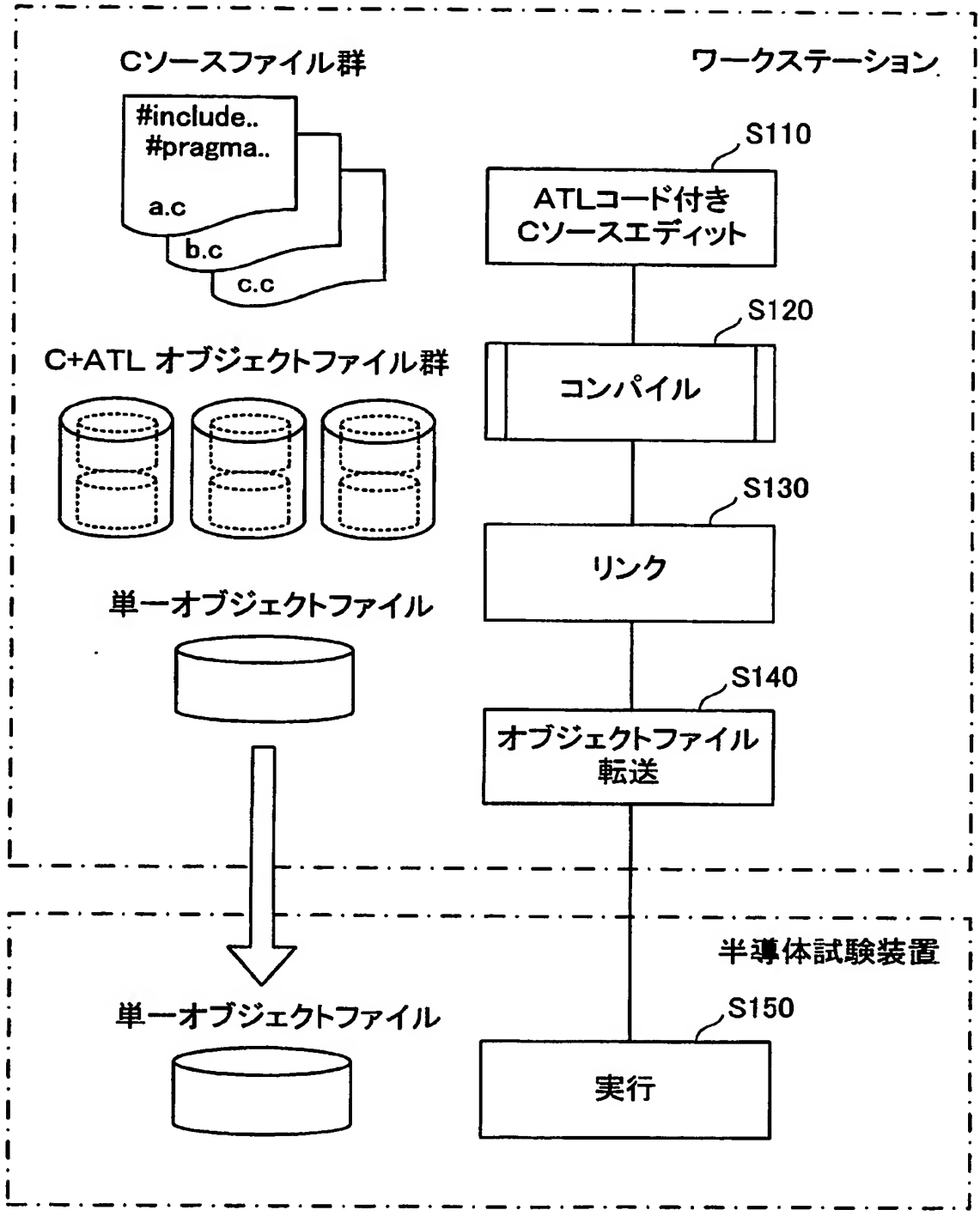
１３１　被測定デバイス

【書類名】　図面

【図１】


10
ワークステーション
20
21
プロセッサ
28
エディタ
統合コンパイラ
34
汎用言語コンパイラ
29
独自仕様言語コンパイラ
30
統合デバッガ
35
汎用言語デバッガ
31
独自仕様言語デバッガ
32
リンカ
33
OSカーネル
GUI処理部
22
通信バスドライバ
23
ハードディスクドライバ
24
マウスドライバ
25
キーボードドライバ
26
ディスプレイドライバ
27
ハードディスク装置
42
マウス
43
キーボード
44
ディスプレイ装置
45
通信インターフェース
41
11
半導体試験装置

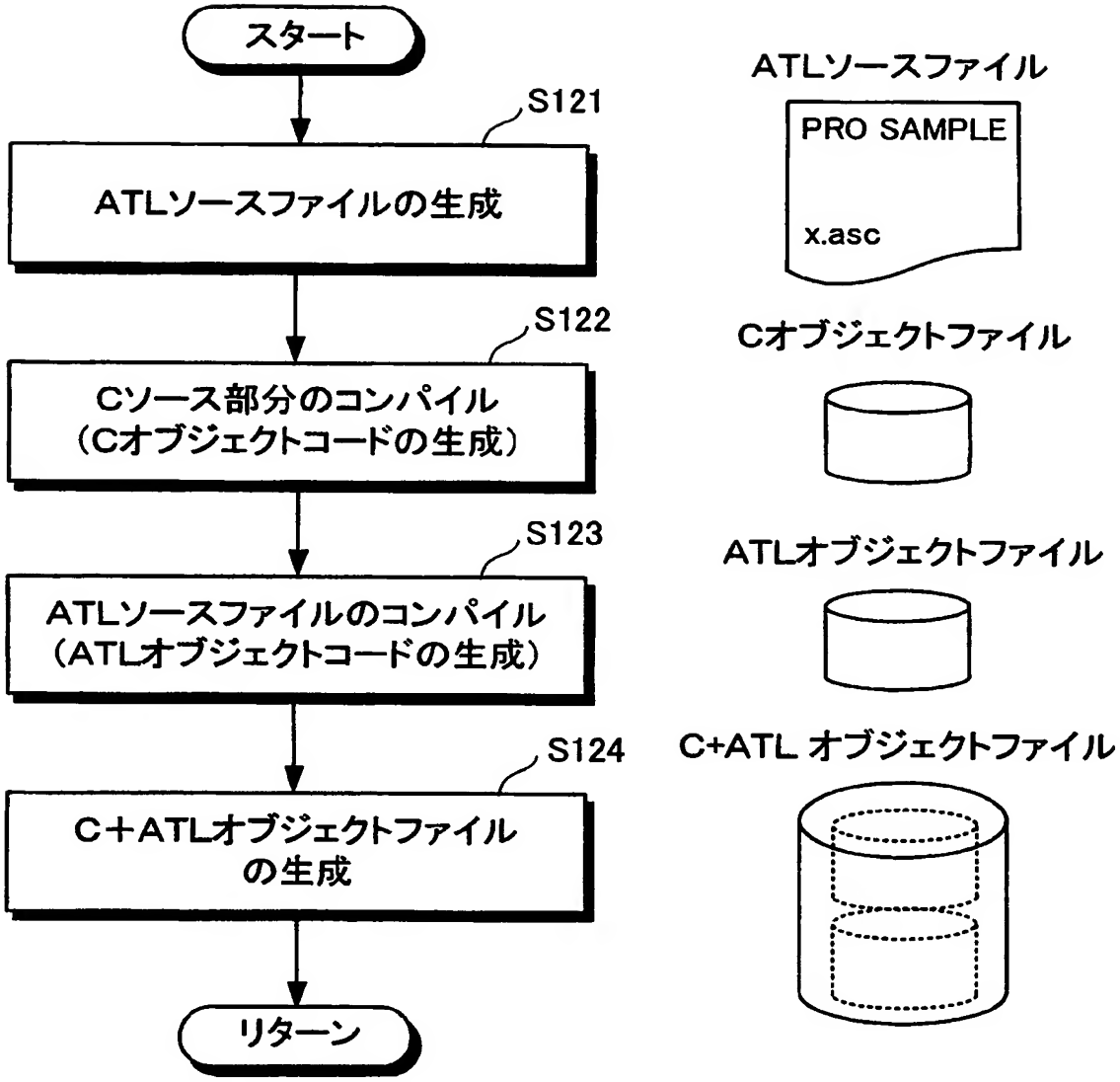
【図２】


Cソースファイル群
#include..
#pragma..
a.c
b.c
c.c
C+ATL オブジェクトファイル群
単一オブジェクトファイル
ワークステーション
S110
ATLコード付き
Cソースエディット
S120
コンパイル
S130
リンク
S140
オブジェクトファイル
転送
半導体試験装置
単一オブジェクトファイル
S150
実行

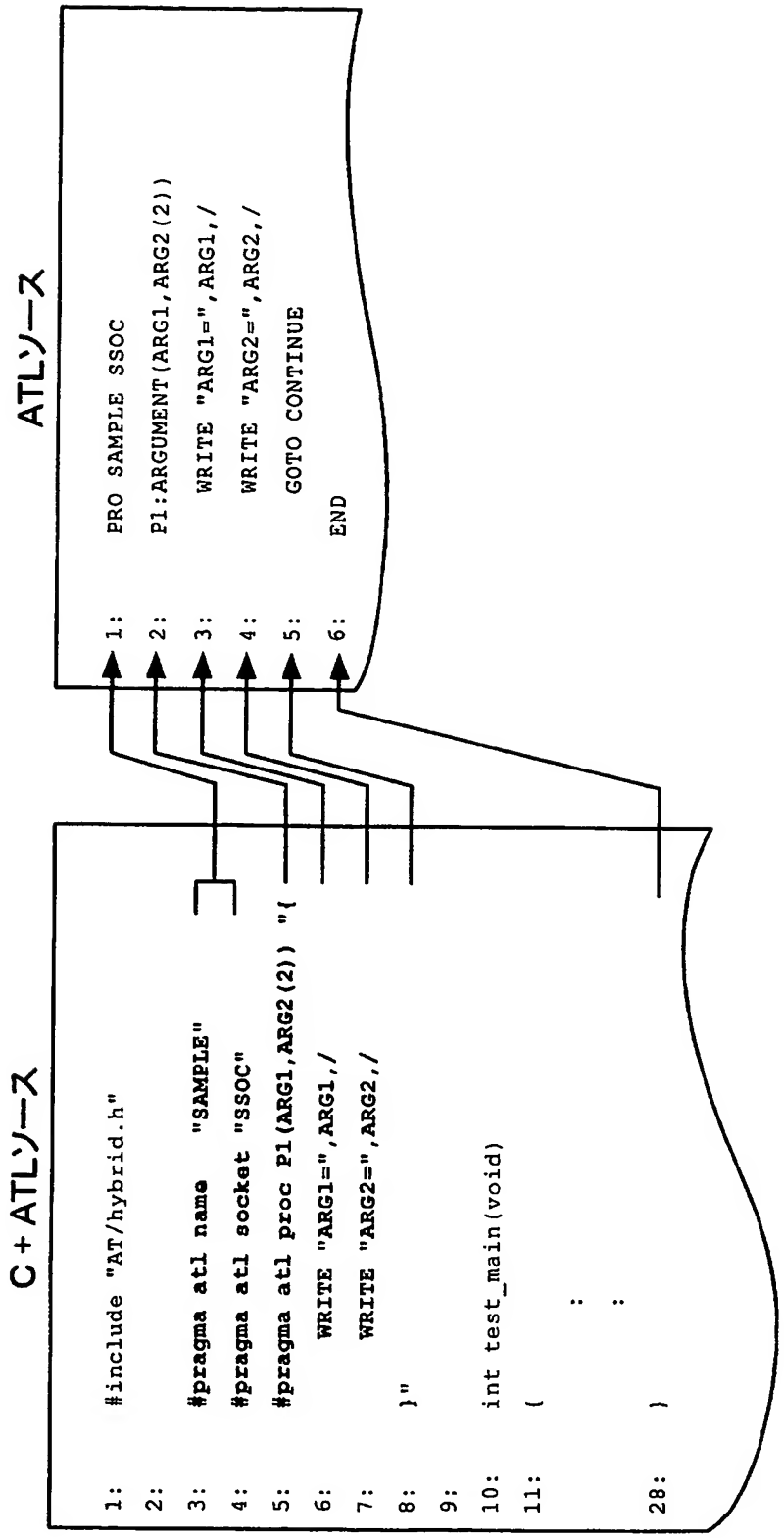
【図３】

```
1:      #include "AT/hybrid.h"
2:
3:      #pragma atl name      "SAMPLE"
4:      #pragma atl socket    "SSOC"
5:      #pragma atl proc P1(ARG1,ARG2(2)) "{
6:           WRITE "ARG1=",ARG1,/
7:           WRITE "ARG2=",ARG2,/
8:      }"
9:
10:     int test_main(void)
11:     {
12:      int idata;
13:      struct atreal rdata[2];
14:
15:      if(ATL_init2(EM_ATL) != 0){
16:         AT_perror("ATL_init2()");
17:             exit(1);
18:      }
19:
20:      idata = 1;
21:      AT_strtorm(rdata, 2, "0.5V", "1.2V");
22:      if(ATL_proc(EM_ATL, "P1", INTE(&idata), DREAL(rdata, 2)) != 0){
23:          AT_perror("P2");
24:          exit(1);
25:      }
26:
27:      return(0);
28:      }
```

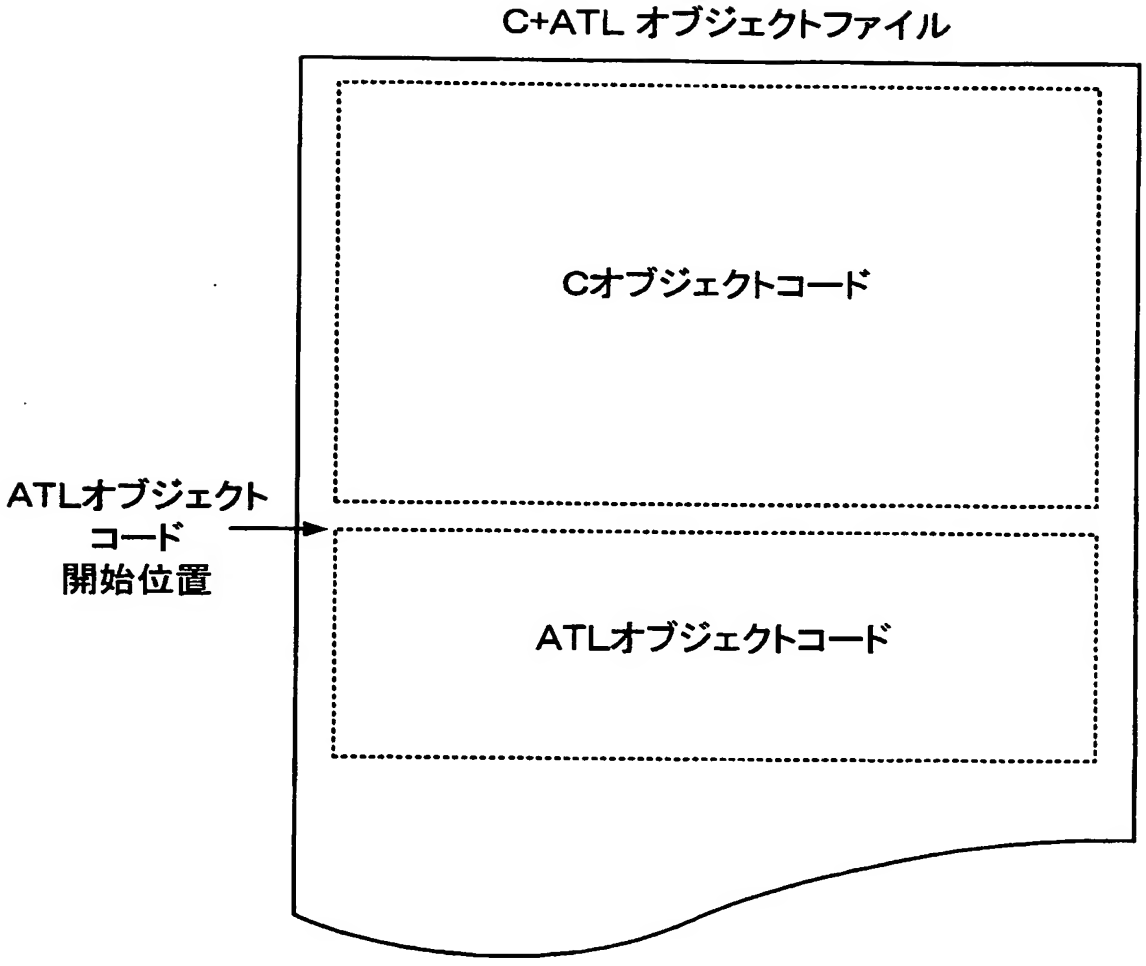
【図４】


スタート
S121
ATLソースファイルの生成
S122
Cソース部分のコンパイル
（Cオブジェクトコードの生成）
S123
ATLソースファイルのコンパイル
（ATLオブジェクトコードの生成）
S124
C＋ATLオブジェクトファイル
の生成
リターン
ATLソースファイル
PRO SAMPLE
x.asc
Cオブジェクトファイル
ATLオブジェクトファイル
C+ATL オブジェクトファイル

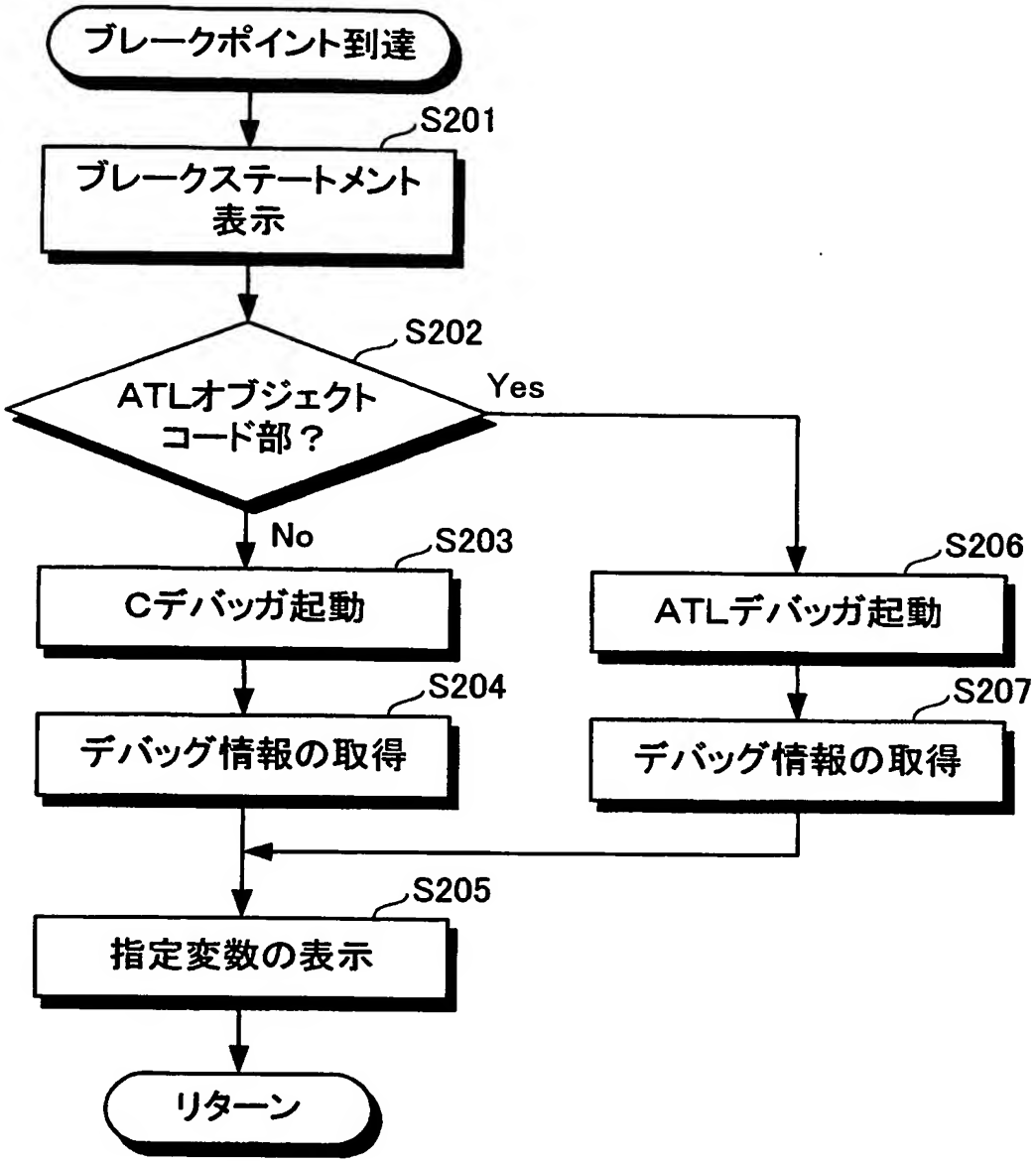
【図５】


ATLソース
1: PRO SAMPLE SSOC
2: P1:ARGUMENT(ARG1,ARG2(2))
3: WRITE "ARG1=",ARG1,/
4: WRITE "ARG2=",ARG2,/
5: GOTO CONTINUE
6: END
C+ATLソース
1: #include "AT/hybrid.h"
2:
3: #pragma atl name "SAMPLE"
4: #pragma atl socket "SSOC"
5: #pragma atl proc P1(ARG1,ARG2(2)) "{
6: WRITE "ARG1=",ARG1,/
7: WRITE "ARG2=",ARG2,/
8: }"
9:
10: int test_main(void)
11: {
:
:
28: }

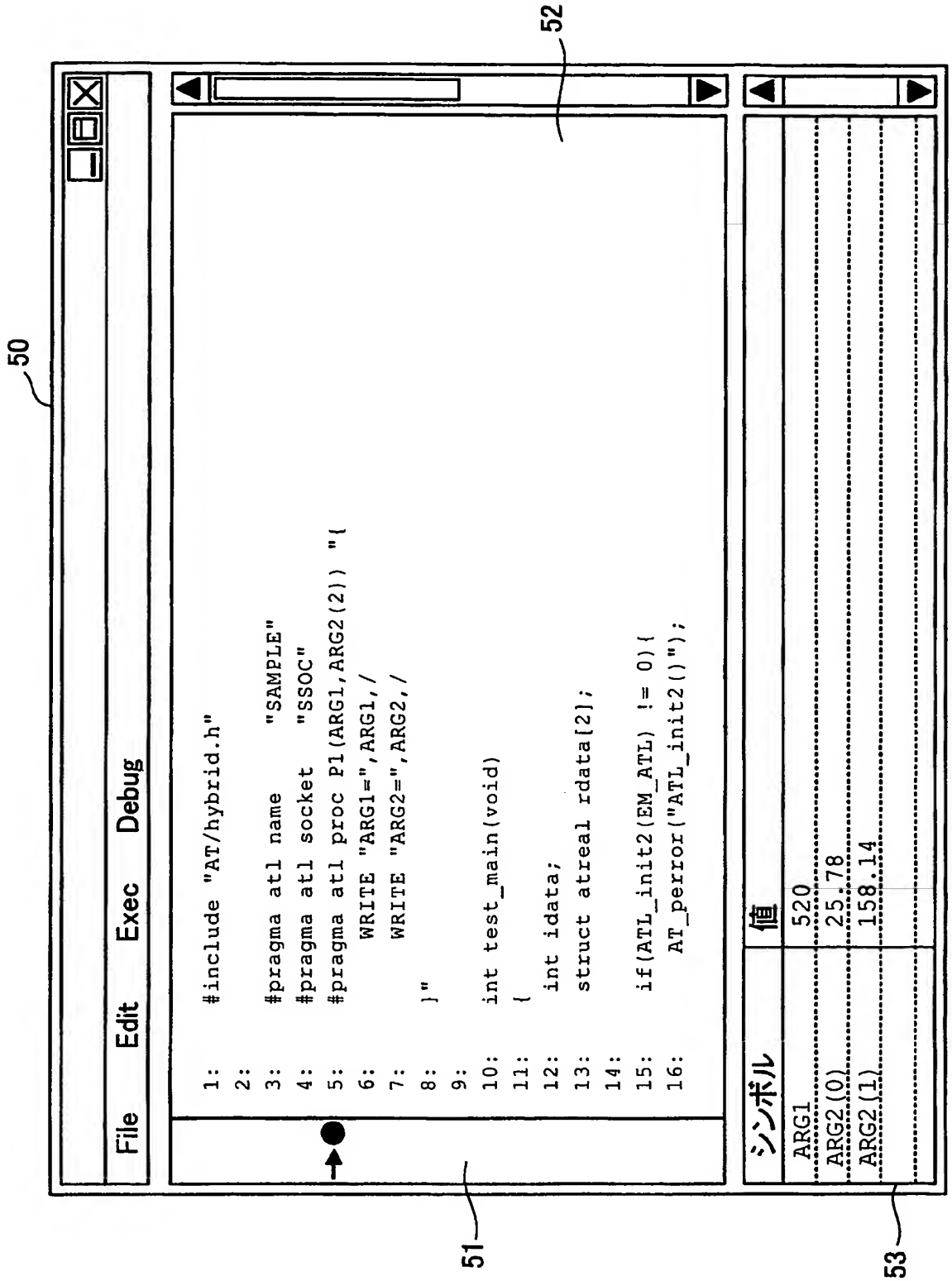
【図６】


C+ATL オブジェクトファイル
Cオブジェクトコード
ATLオブジェクト
コード
開始位置
ATLオブジェクトコード

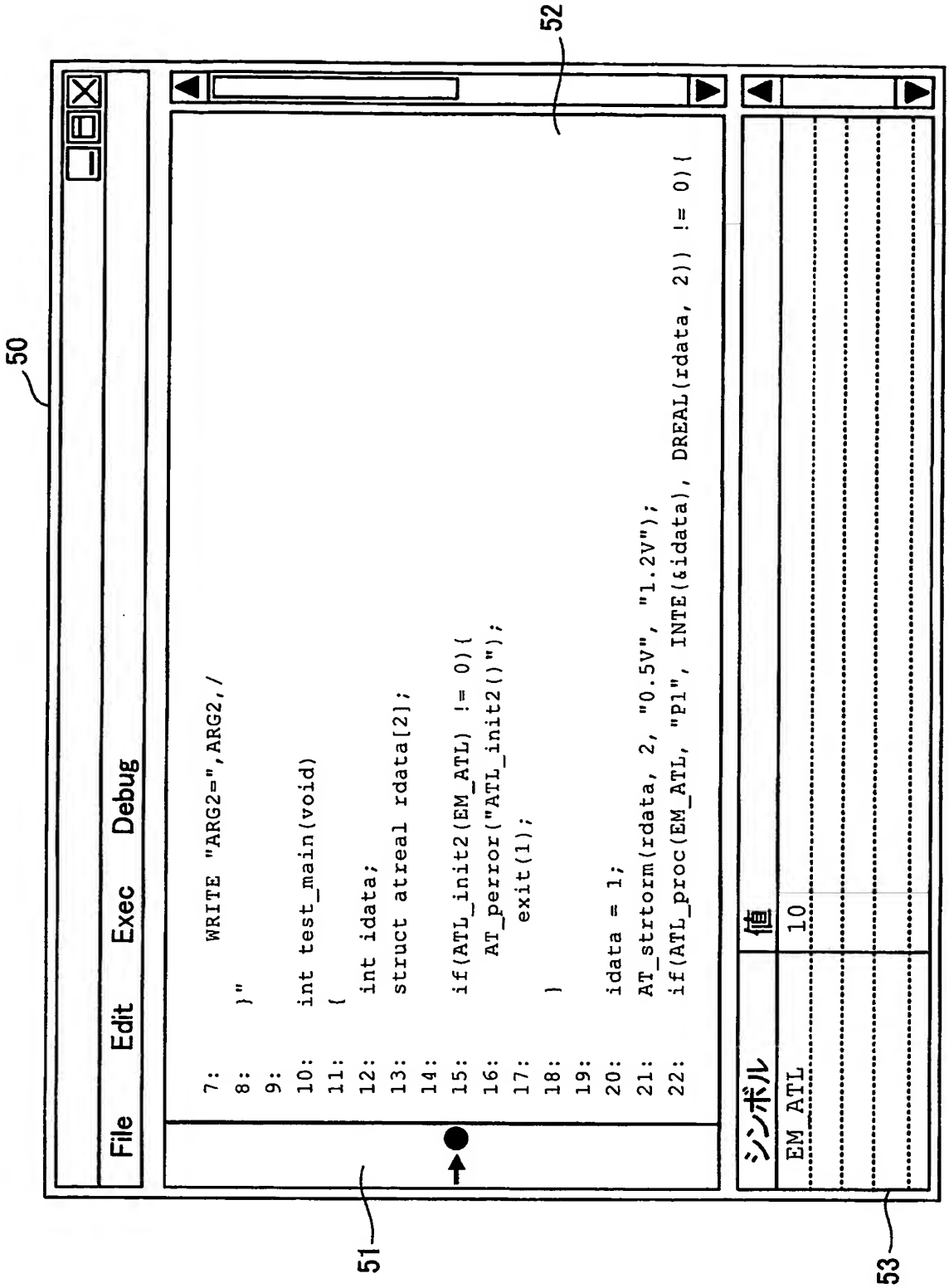
【図７】


ブレークポイント到達
S201
ブレークステートメント表示
S202
ATLオブジェクトコード部？
Yes
No
S203
Cデバッガ起動
S204
デバッグ情報の取得
S206
ATLデバッガ起動
S207
デバッグ情報の取得
S205
指定変数の表示
リターン

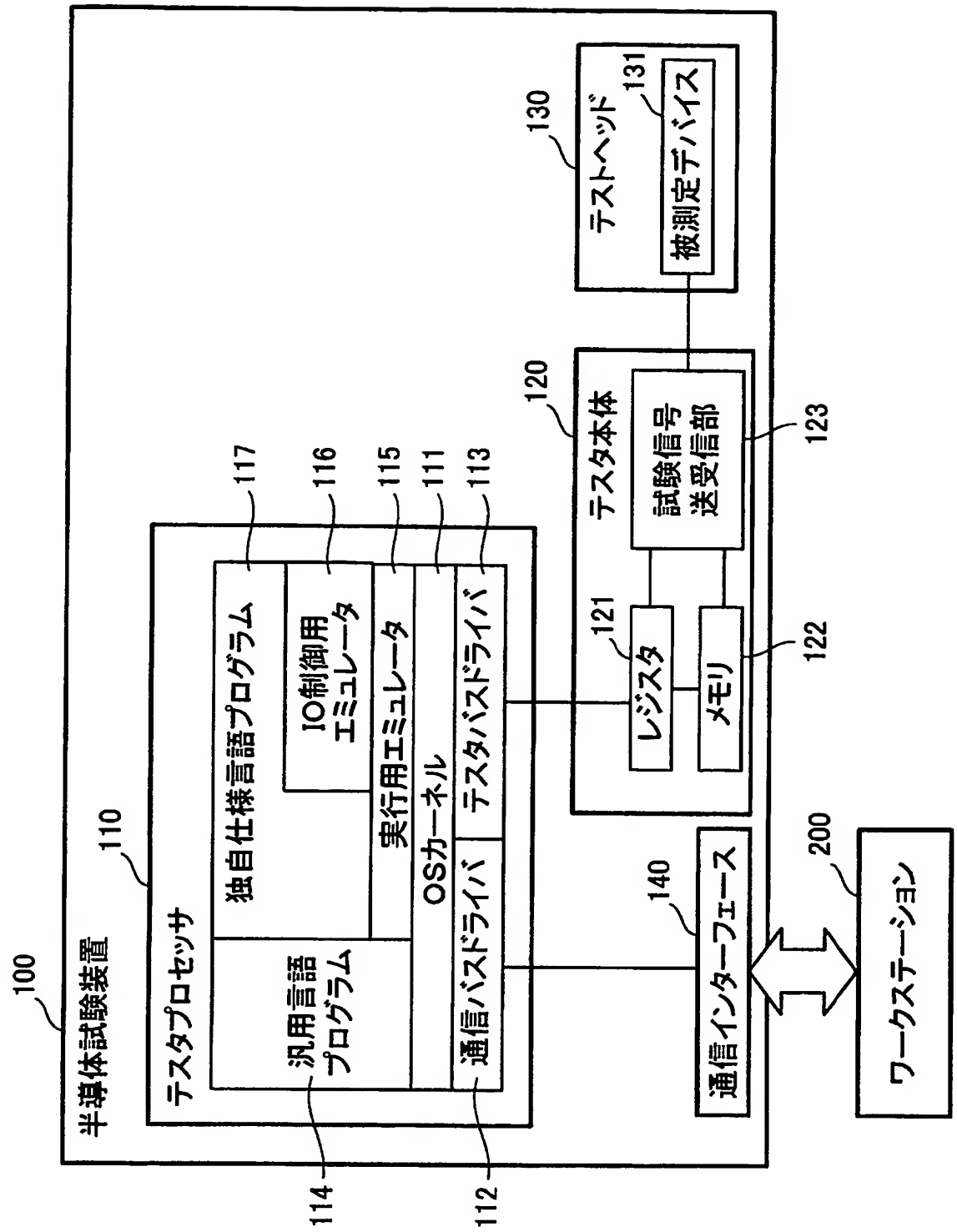
【図８】


50
52
File
Edit
Exec
Debug
1: #include "AT/hybrid.h"
2:
3: #pragma atl name "SAMPLE"
4: #pragma atl socket "SSOC"
5: #pragma atl proc P1(ARG1,ARG2(2)) "{
6: WRITE "ARG1=",ARG1,/
7: WRITE "ARG2=",ARG2,/
8: }"
9:
10: int test_main(void)
11: {
12: int idata;
13: struct atreal rdata[2];
14:
15: if(ATL_init2(EM_ATL) != 0){
16: AT_perror("ATL_init2()");
シンボル
値
ARG1
520
ARG2(0)
25.78
ARG2(1)
158.14
51
53

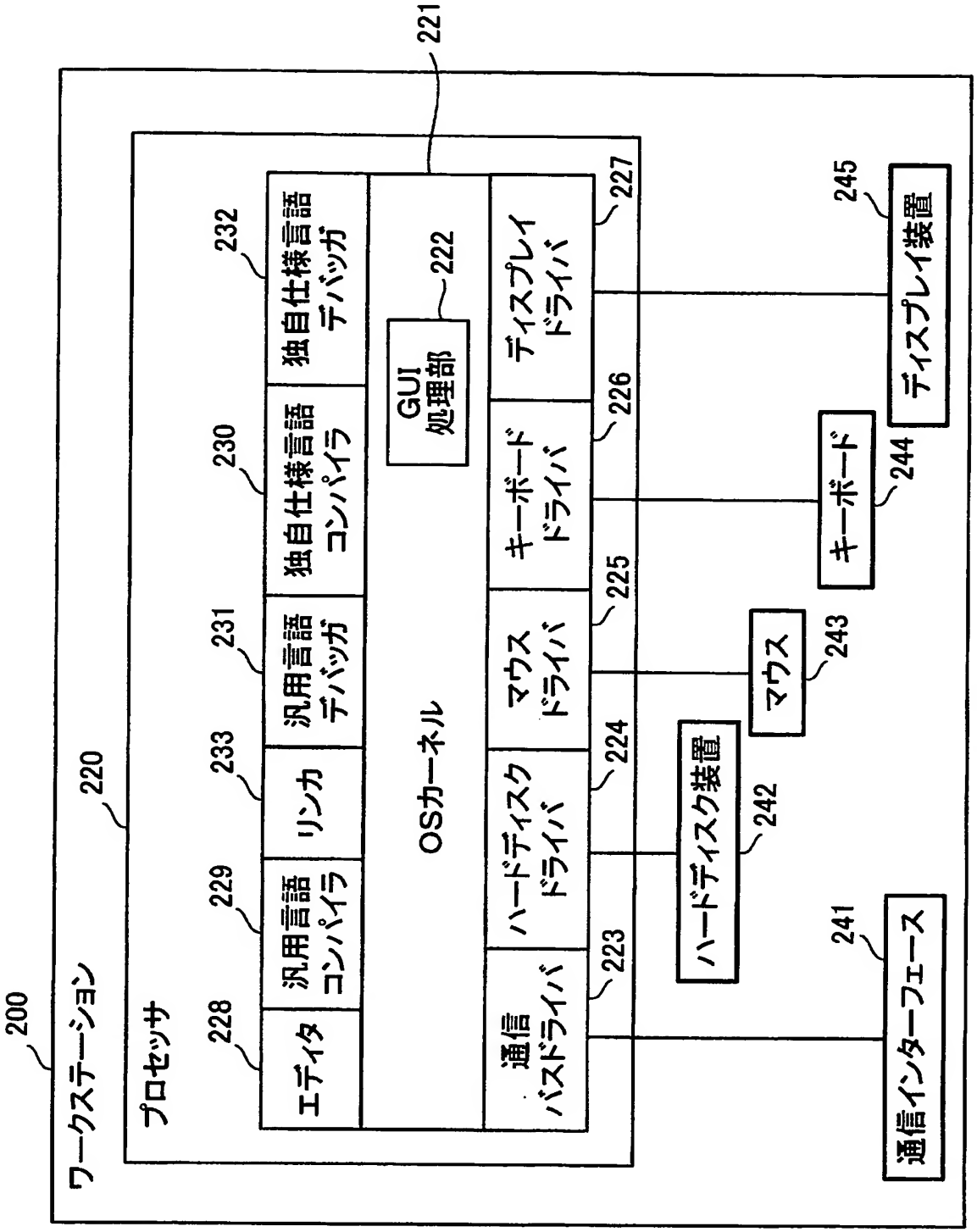
【図９】


50
51
52
53
File
Edit
Exec
Debug
7: WRITE "ARG2=",ARG2,/
8:)"
9:
10: int test_main(void)
11: {
12: int idata;
13: struct atreal rdata[2];
14:
15: if(ATL_init2(EM_ATL) != 0){
16: AT_perror("ATL_init2()");
17: exit(1);
18: }
19:
20: idata = 1;
21: AT_strtorm(rdata, 2, "0.5V", "1.2V");
22: if(ATL_proc(EM_ATL, "P1", INTE(&idata), DREAL(rdata, 2)) != 0){
シンボル
値
EM ATL
10

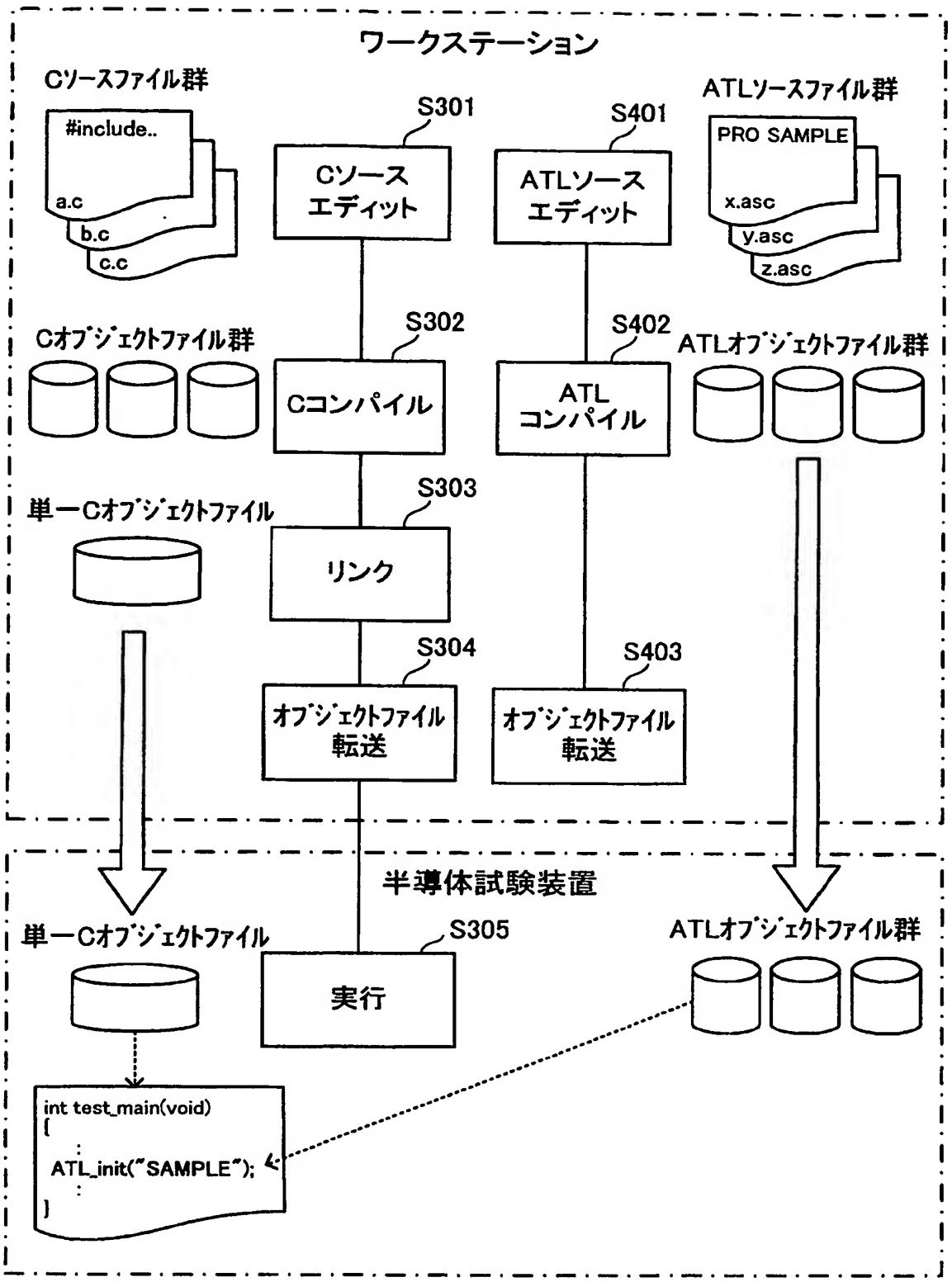
【図１０】


半導体試験装置
100
テスタプロセッサ
110
独自仕様言語プログラム
117
IO制御用エミュレータ
116
実行用エミュレータ
115
OSカーネル
111
テスタバスドライバ
113
汎用言語プログラム
114
通信バスドライバ
112
テスタ本体
120
レジスタ
121
メモリ
122
試験信号送受信部
123
テストヘッド
130
被測定デバイス
131
通信インターフェース
140
ワークステーション
200

【図１１】


200
ワークステーション
220
プロセッサ
221
OSカーネル
222
GUI
処理部
228
エディタ
229
汎用言語
コンパイラ
233
リンカ
231
汎用言語
デバッガ
230
独自仕様言語
コンパイラ
232
独自仕様言語
デバッガ
223
通信
バスドライバ
224
ハードディスク
ドライバ
225
マウス
ドライバ
226
キーボード
ドライバ
227
ディスプレイ
ドライバ
241
通信インターフェース
242
ハードディスク装置
243
マウス
244
キーボード
245
ディスプレイ装置

【図１２】


ワークステーション
Cソースファイル群
#include..
a.c
b.c
c.c
S301
Cソース
エディット
S401
ATLソース
エディット
ATLソースファイル群
PRO SAMPLE
x.asc
y.asc
z.asc
Cオブジェクトファイル群
S302
Cコンパイル
S402
ATL
コンパイル
ATLオブジェクトファイル群
単一Cオブジェクトファイル
S303
リンク
S304
オブジェクトファイル
転送
S403
オブジェクトファイル
転送
半導体試験装置
単一Cオブジェクトファイル
S305
実行
ATLオブジェクトファイル群
int test_main(void)
{
ATL_init("SAMPLE");
}

【書類名】　要約書

【要約】

【課題】　異種言語のプログラムが実行される環境において、開発時および実行時に必要なプログラムファイルの数を減少させることのできるプログラム開発支援装置、プログラム実行装置、コンパイル方法およびデバッグ方法を提供する。

【解決手段】　ワークステーション（プログラム開発装置に相当）上で、汎用言語ソースプログラム（図ではＣ言語）のプリプロセッサ記述部に、独自仕様言語ソースプログラム（図ではＡＴＬ）の記述を埋め込んだ異種言語混在ソースプログラムを作成する（ステップＳ１１０）。そして、その異種言語混在ソースプログラムから汎用言語ソースプログラム記述部と独自仕様言語ソースプログラム記述部を取り出して、それぞれのコンパイラでコンパイルし、得られたそれぞれのオブジェクトコードを結合して、一つのオブジェクトファイルを生成する（ステップＳ１２０）。

【選択図】　図２

特願２００２－３０５０１０

出　願　人　履　歴　情　報

識別番号　　［３９０００５１７５］

１．変更年月日　　１９９０年１０月１５日
［変更理由］　　新規登録
住　所　　東京都練馬区旭町１丁目３２番１号
氏　名　　株式会社アドバンテスト